`EloChoice (v. 0.28)` - a brief tutorial

Christof Neumann and Andrew P. Clark

October 11, 2015

Contents

1 introduction

The overall goal of this document is to provide a short manual on how to calculate Elo-ratings for attractiveness ratings on pairwise presented stimuli. Below, we provide a worked example (section 2) based on an artificially generated data set. We suggest you go through this example before using the method on your own data set. Section 3 explains in a bit more detail the suggested reliability index as means to evaluate stability in ratings. The next section (section 4) deals with an empirical data set, which comes with the package.

In general, the underlying idea of the Elo-rating procedure is to update individual scores after pair-wise contests based on the expected outcome of the contest *before* a contest actually takes place. The more expected an outcome was, the smaller is the change in scores of the two contestants. Conversely, the more unexpected an outcome was, the larger are the score changes. The expectation of an contest outcome is expressed as the difference in Elo-ratings before the contest. Inherent in this general philosophy is that scores of contestants change over time - think of a chess or tennis player or an animal that at the start of her career will have a relatively low score (and/or rank), which may increase over time and eventually will drop again.

Now, when we think of a pair of visual stimuli as 'contestants', for example in the context of attractiveness ratings, the aspect of dynamics over time is much less important. Typically, an experiment of attractiveness is conducted over a very brief period of time (at least relatively, compared to a chess player's career or a monkey's life), and as such we expect such changes in attractiveness over time to play only a negligible role. Elo-rating, as implemented in this package and tutorial, will still result in a reliable ranking of stimuli attractiveness. The crucial aspect of this package is that Elo-ratings are based on multiple randomized sequences of ratings, which we refer to as *mElo* in the accompanying paper. Though not strictly necessary, we think this is a prudent approach, because the order in which rating trials occur in the sequence may affect the

final Elo-ratings of the stimuli, at least in small data sets (small in terms of stimuli or small in number of rating trials).

2 a worked example

The first thing to be done is to install and load the package.[1] Eventually, a simple `install.packages("EloChoice")` will suffice (once the package is on the official R server). Also note that we need the packages `Rcpp` and `RcppArmadillo` installed, which should be automatically downloaded and installed if you use the `install.packages("EloChoice")` command.[2]

```
install.packages(EloChoice) # install package (to be done once)

library(EloChoice) # load package (every time you want to use the package)
```

2.1 get the data into R

If you already know how to read your raw data into R, you can skip this section. We assume that you have your ratings organized in a table, in which each line corresponds to a rating event/trial. There is at least a column for the stimulus that was preferred by the rater and the one that was not. Additional columns are likely to be present and table 1 provides an example. Most likely you will have organized this table in a spreadsheet software like Excel or OpenOffice. The perhaps simplest way of reading a data set into R is to first save your data table from the spreadsheet software as tab-delimited text file (File>Save as...> and then choose "Tab Delimited Text (.txt)").

Having such a text-file, it is then easy to read that data set into R and save it as an object named `xdata`[3]:

```
xdata <- read.table("c:\\datafiles\\myfile.txt", sep="\t", header=T) # Windows
xdata <- read.table("/Volumes/mydrive/myfile.txt", sep="\t", header=T) # Mac
str(xdata)
```

Note that you have to specify the full path to the place where the data file is saved.[4] The `sep="\t"` argument tells R that you used a tab to separate entries within a line. The `header=T` argument tells R that the first line in your table are column headers. Finally, the `str(xdata)` command gives a brief overview over the data set, which you can use to check whether the import went smoothly (for example, as indicated by the number of lines (`obs.` in the output of `str()`), which should correspond to the number of trials in your data set).

Table 1: A possible data set layout. Note that R replaces space in column names with periods.

preferred stimulus	losing stimulus	rater	date	time
ab	cf	A	2010-01-01	14:34:01
cf	xs	A	2010-01-01	14:34:08
ab	xs	A	2010-01-01	14:34:11
dd	cf	A	2010-01-01	14:34:15
ab	cf	B	2010-01-04	09:17:20

If the import was successful, the only thing you need to know is that for the functions of the package to work we need to access the columns of the data table individually. This can be achieved by using the dollar character. For example `xdata$preferred.stimulus` returns the column with the preferred stimuli (as per table 1). If you are unfamiliar with this, it will become clear while going through the examples in the next section.

[1] Installing has to be done once, while loading the package has to be done each time you restart `R`

[2] `install.packages("Rcpp")` and `install.packages("RcppArmadillo")` will install the two packages by hand

[3] you can name this object any way you like, `xdata` is just a personal convention

[4] You can also use `setwd()` to define a new root directory or you can use (the freely available) RStudio, which offers nice options to work in projects that facilitate reading of files among others.

2.2 random data

Throughout this first part of the tutorial, we will use randomly generated data sets. We begin by creating such a random data set with the function `randompairs()`, which we name `xdata`.[5]

```
set.seed(123)
xdata <- randompairs(nstim = 20, nint = 500, reverse = 0.1)
head(xdata)
```

```
  index winner loser
1     1      p     x
2     2      t     v
3     3      r     s
4     4      n     z
5     5      w     t
6     6      k     u
```

The command `set.seed(123)` simply ensures that each time you run this it will create the same data set as it shown in this tutorial. If you want to create a truly random data set, just leave out this line.

The function as is created a data set with 20 different stimuli (`nstim=20`), which were presented in 500 presentations/trials (`nint=500`). The `head()` command displays the first six lines of the newly created data set.[6] You can see the stimulus IDs: the `winner`-column refers to the preferred stimulus (the 'winner' of the trial) and the `loser`-column to the second/unpreferred stimulus (the 'loser'). The `index`-column simply displays the original order in which the stimuli were presented. Note also that the data set is generated in a way such that there is always a preference for the stimulus that comes first in alphanumeric order, which is reversed in 10% of trials (`reverse=0.1`). This ensures that a hierarchy of stimuli actually arises.

2.3 Calculating the ratings

We then can go on to calculate the actual ratings from this sequence. Again, to enable you to obtain the exact results as presented in this tutorial, I use the `set.seed()` function.

```
set.seed(123)
res <- elochoice(winner = xdata$winner, loser = xdata$loser, runs = 500)
summary(res)
```

```
Elo ratings from 20 stimuli
total (mean/median) number of rating events: 500 (50/50)
range of rating events per stimulus: 33 - 61
startvalue: 0
k: 100
randomizations: 500
```

The two code pieces `winner = xdata$winner` and `loser = xdata$loser` specify the two columns in our data table that represent the winning ('preferred') and losing ('not preferred') stimuli, respectively. The `runs = 500` bit indicates how many random sequences of presentation order we want to calculate.

[5] You are by no means bound to use the name `xdata` for this object, this is just a personal convention

[6] If you want to see the entire data set, simply type `xdata`

We saved the results in an object named `res`, from which can we can get a brief summary with the `summary(res)` function. From this, we can see that there were 20 stimuli in the data set, each appearing between 33 and 61 times. Also note that we randomized the original sequence 500 times. In case you have larger data sets, you may want to reduce the number of randomizations to reduce the computation time and to explore whether everything runs as it should.[7]

Next, we obviously want to see what the actual Elo-ratings are for the stimuli used in the data set. For this, we use the function `ratings()`.

```
ratings(res, show="original", drawplot=FALSE)
```

```
  a   h   g   b   j   c   n   k   o   p   r   s   u   q   w   v
423 376 360 312 262 236 221  76  25  17  -6 -88 -123 -124 -146 -202
  t   y   x   z
-264 -337 -470 -548
```

The `show="original"` argument specifies that we wish to see the ratings as obtained from the initial (original) data sequence. If we want to have the ratings averaged across all randomizations ('mElo'), we change the argument to `show="mean"`.

```
ratings(res, show="mean", drawplot=FALSE)
```

```
      a       h       b       c       g       j       k       o
450.486 386.924 360.402 313.614 240.978 231.416 167.092 103.174
      n       p       q       r       s       t       w       u
 96.060  49.672 -39.202 -103.966 -112.190 -174.578 -185.730 -214.558
      v       y       x       z
-276.592 -322.028 -441.000 -529.974
```

Similarly, you can also request *all* ratings from *all* randomizations, using `show="all"` or return the ranges across all randomizations with `show="range"`.

If you wish to export ratings, you can use the `write.table()` function, which saves your data in a text file that can easily be opened in a spreadsheet program. First, we save the rating results into a new object, `myratings`, which we then export. Note that the resulting text file will be in a 'long' format, i.e. each stimulus along with its original or mean rating will appear as one row.

```
myratings <- ratings(res, show="mean", drawplot=FALSE)
# Windows
xdata <- write.table(myratings, "c:\\datafiles\\myratings.txt", sep="\t",
                     header=T)
# Mac
xdata <- write.table(myratings, "/Volumes/mydrive/myratings.txt", sep="\t",
                     header=T)
```

If you want to export the ratings from each single randomization (i.e. `show="all"`) or the range of ratings across all randomizations (`show="range"`), the layout of the text file will be 'wide', i.e. each stimulus appears as its own column with each row representing ratings after one randomization or two rows representing the minimum and maximum rating values. By default, R appends row names to the text output, which is not convenient in this case so we turn this option off with `row.names=F`.

[7] On my laptop PC, the calculations of this example take less than a second, but this can drastically increase if you have larger data sets and/or want to use larger number of randomizations.

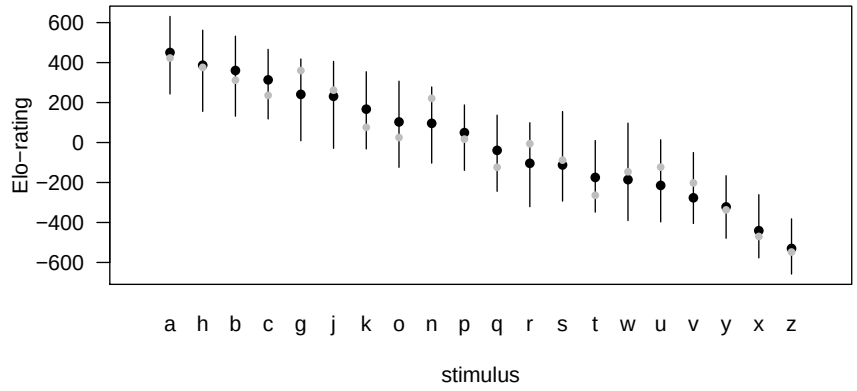

Figure 1: Elo ratings of 20 stimuli after 500 rating events and 500 randomizations of the sequence. The black circles represent the average rating at the end of the 500 generated sequences for each stimulus, and the black lines represent their ranges. The grey circles show the final ratings from the original sequence.

```
myratings <- ratings(res, show="all", drawplot=FALSE)
# Windows
xdata <- write.table(myratings, "c:\\datafiles\\myratings.txt", sep="\t",
                     header=T, row.names=F)
# Mac
xdata <- write.table(myratings, "/Volumes/mydrive/myratings.txt", sep="\t",
                     header=T, row.names=F)
```

Finally, the `ratings()` function also allows you to take a first graphical glance at how the randomizations affected the ratings.

```
ratings(res, show=NULL, drawplot=TRUE)
```

Figure 1 shows the average ratings across the 500 randomizations as black circles, while the ratings from the original/initial sequence are indicated by the smaller red circles The vertical bars represent the ranges of Elo-ratings across the 500 randomizations for each stimulus.

3 reliability-index

3.1 background

We define an reliability-index as $R = 1 - \frac{\sum u}{N}$, where N is the total number of rating events/trials for which an expectation for the outcome of the trial existed[8] and u is a vector containing 0's and 1's, in which a 0 indicates that the preference in this trial was according to the expectation (i.e. the stimulus with the higher Elo-rating before the trial was preferred), and a 1 indicates a trial in which the expectation was violated, i.e. the stimulus with the lower Elo-rating before the trial was preferred (an 'upset'). In other words, R is the proportion of trials that went in accordance with the expectation. Note that trials without any expectation, i.e. those for which ratings for both stimuli are identical, are excluded from the calculation. Subtracting the proportion from 1 is

[8]Consequently, the maximum value N can take is the number of trials minus one, because for at least the very first trial in a sequence, no expectation can be expressed

done to ensure that if there are no upsets ($\sum u = 0$), the index is 1, thereby indicating complete agreement between expectation and observed rating events. For example, in table 2 there are ten rating events/trials, four of which go against the expectation (i.e. they are upsets), yielding an reliability-index of $1 - 4/10 = 0.6$.

This approach can be extended to calculate a weighted reliability index, where the weight is given by the absolute Elo-rating difference, an index we denote R' and which is defined as $R' = 1 - \sum_{i=1}^{N} \frac{u_i * w_i}{\sum w}$, where u_i is the same vector of 0's and 1's as described above and w_i is the absolute Elo-rating difference, i.e. the weight. Following this logic, stronger violations of the expectation contribute stronger to the reliability index than smaller violations. For example, column 3 in table 2 contains fictional rating differences, which for illustrative purposes are assigned in a way such that the four largest rating differences (200, 200, 280, 300) correspond to the four upsets, which should lead to a smaller reliability index as compared to the simpler version described earlier. Applying this leads to $R' = 1 - 0.57 = 0.43$. In contrast, if we apply the smallest rating differences (90, 100, 120, 140, as per column 4 in table 2) to the upsets, this should lead to a larger reliability-index, which it does: $R' = 1 - 0.26 = 0.74$.

Table 2: 10 rating decisions that were either in accordance with the prediction or not. Two different rating differences are given to illustrate the weighted upset index. Note that the values are the same, just their assignment to different interactions is changed and consequently the column means are the same for both (173).

higher rated = preferred	upset	rating difference 1	rating difference 2
1	yes	200	90
1	yes	300	100
0	no	100	300
0	no	150	280
1	yes	200	120
0	no	140	200
1	yes	280	140
0	no	90	150
0	no	150	200
0	no	120	150

3.2 application

To calculate the reliability-index, we use the function `reliability()`. Note that we calculated our initial Elo-ratings based on 500 randomizations, so to save space, I'll display only the first six lines of the results (the `head(...)` function).

```
upsets <- reliability(res)
head(upsets)
```

```
      upset upset.wgt totIA
1 0.8154158 0.8749464   493
2 0.8174442 0.8721155   493
3 0.7991886 0.8731617   493
4 0.8056680 0.8732986   494
5 0.8185484 0.8769738   496
6 0.8165323 0.8686285   496
```

Each line in this table represents one randomization.[9] The first column represents the unweighted and the second the weighted reliability index (R and R'), which is followed by the total number of trials that contributed to the calculation of the index. Note that this number cannot

[9] the first line corresponds to the actual original sequence

reach 500 (our total number of trials in the data set) because at least for the very first trial we did not have an expectation for the outcome of that trial.

We then calculate the average values for both the unweighted and weighted upset indices.

```
mean(upsets$upset)
```

```
[1] 0.811623
```

```
mean(upsets$upset.wgt)
```

```
[1] 0.87683
```

Remember that our data set contained a fairly low number of 'reversals', i.e. 10% of trials went against the predefined preference (e.g. 'A' is preferred over 'K'). As such, the R and R' values are fairly high ($\sim 0.8 - 0.9$). If we create another data set, in which reversals are more common, we can see that the values go down.

```
set.seed(123)
xdata <- randompairs(nstim = 20, nint = 500, reverse = 0.3)
res <- elochoice(winner = xdata$winner, loser = xdata$loser, runs = 500)
upsets <- reliability(res)
mean(upsets$upset)
```

```
[1] 0.6009882
```

```
mean(upsets$upset.wgt)
```

```
[1] 0.6462721
```

4 an empirical example

In the following, we go through an empirical example data set. Here, 56 participants were asked to choose the one out of two presented bodies which depicted the stronger looking male. Each of the 82 stimuli appeared 112 times, resulting in a total of 4,592 rating trials.

We start by loading the data set. Then we calculate the Elo-ratings, show the average ratings and plot them (figure 2).

```
data(physical)
set.seed(123)
res <- elochoice(winner = physical$Winner, loser = physical$Loser, runs = 500)
summary(res)
```

```
Elo ratings from 82 stimuli
total (mean/median) number of rating events: 4592 (112/112)
range of rating events per stimulus: 112 - 112
startvalue: 0
k: 100
randomizations: 500
```

```
ratings(res, show = "mean", drawplot = FALSE)
```

```
    P012     P070     P076     P142     P168     P013     P169     P150
 634.890  594.636  564.144  564.084  489.200  454.752  444.490  438.770
    P060     P143     P161     P017     P016     P061     P003     P027
 420.564  411.304  380.524  349.902  346.874  330.388  320.228  286.242
    P002     P046     P148     P006     P103     P007     P038     P075
```

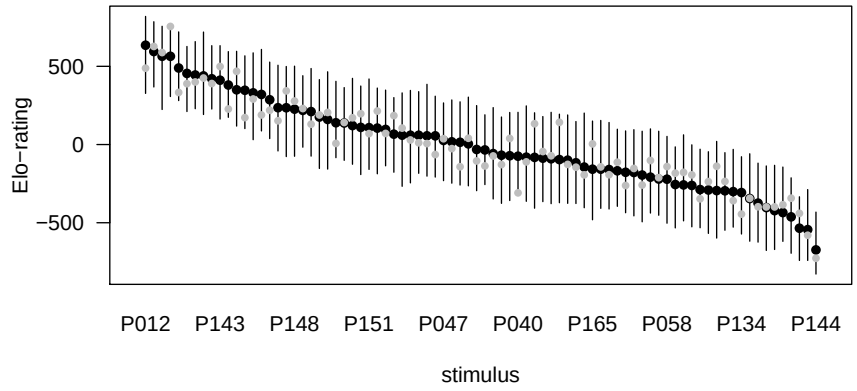

Figure 2: Elo ratings of 82 stimuli after 4,592 rating events and 500 randomizations of the sequence. The black circles represent the average rating at the end of the 500 generated sequences, and the black lines represent their ranges. The grey circles show the final ratings from the original sequence. Note that not all stimulus IDs fit on the x-axis, so most are omitted.

```
 235.422  235.164  226.182  218.364  210.590  175.764  160.536  139.222
    P089     P119     P053     P151     P147     P050     P126     P129
 137.570  121.326  109.632  108.052  105.076   96.488   66.296   59.824
    P044     P020     P155     P008     P047     P117     P014     P004
  59.592   59.228   55.650   55.564   26.126   17.458   14.410    4.172
    P078     P166     P092     P112     P083     P040     P034     P121
 -31.650  -34.438  -57.890  -68.312  -71.590  -74.642  -80.666  -82.202
    P021     P110     P163     P132     P015     P090     P165     P036
 -87.140  -91.170  -96.086  -99.912 -116.514 -144.022 -157.626 -157.668
    P171     P124     P018     P057     P077     P079     P125     P058
-159.726 -167.988 -177.456 -178.644 -195.106 -208.544 -220.562 -221.670
    P031     P127     P128     P052     P098     P123     P172     P029
-255.682 -258.202 -261.508 -287.894 -290.642 -294.466 -295.756 -300.986
    P134     P086     P019     P170     P140     P080     P042     P159
-307.024 -345.188 -374.590 -402.974 -422.570 -435.016 -462.394 -534.958
    P085     P144
-544.016 -673.610
```

```
ratings(res, show=NULL, drawplot=TRUE)
```

5 self-contests

Depending on how the stimulus presentation is prepared, there may be cases (trials) in the final data set in which a stimulus is paired with itself ('self-contest'). As long as the presentation is indeed of pairs of stimuli this is not a problem because such self-contests are irrelevant to refine the true rating of a stimulus (as opposed to pairs of two different stimuli). If there are three or more stimuli presented in one trial, the situation becomes different if for example two 'A's are presented with one 'B'. However, this is an issue to be dealt with later, when presentation of triplets or more stimuli is properly implemented in the package (currently under development).

For now, self-contests are excluded from analysis of stimulus pairs for the reason mentioned above. However, a message that such self-contests occur in the data will be displayed in such

cases.[10]

```
 # total of seven trials with two 'self-trials' (trials 6 and 7)
 w <- c(letters[1:5], "a", "b"); l <- c(letters[2:6], "a", "b")
 res <- elochoice(w, l)
 ratings(res, drawplot=FALSE)

  a   b   c   d   e   f
 50   7   1   0   0 -58

 summary(res)

Elo ratings from 6 stimuli
total (mean/median) number of rating events: 5 (1.67/2)
range of rating events per stimulus: 1 - 2
startvalue: 0
k: 100
randomizations: 1

 # total of five trials without 'self-trials'
 w <- c(letters[1:5]); l <- c(letters[2:6])
 res <- elochoice(w, l)
 ratings(res, drawplot=FALSE)

  a   b   c   d   e   f
 50   7   1   0   0 -58

 summary(res)

Elo ratings from 6 stimuli
total (mean/median) number of rating events: 5 (1.67/2)
range of rating events per stimulus: 1 - 2
startvalue: 0
k: 100
randomizations: 1
```

[10] In this document the actual message does not show, but if you run the code yourself you should be able to see it.